**The Report Committee for Cristina Fuentes-Curiel**
**Certifies that this is the approved version of the following report**

# SmartPark: An Intelligent and Dynamic Parking System

**APPROVED BY**
**SUPERVISING COMMITTEE:**

**Supervisor:** _____
Christine Julien

_____
William Bard

# SmartPark: An Intelligent and Dynamic Parking System

**by**

**Cristina Fuentes-Curiel, B.S.**

**Report**

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

**Master of Science in Engineering**

**The University of Texas at Austin**

**August 2013**

For my mother.

## Acknowledgements

I would like to recognize the University of Texas faculty and staff for their support and dedication to making their graduate program such a pleasant and rewarding experience. I would like to thank my supervisor, Professor Christine Julien, for her guidance during the development of this project. I would also like to thank Professor William Bard, for reading this paper and for all the knowledge I gained from his courses during my two years in this program. And finally, I would like to thank my friend, Rodolfo Rosas, for bringing the original SmartPark project to life.

# Abstract

## SmartPark: An Intelligent and Dynamic Parking System

Cristina Fuentes-Curiel, M.S.E.

The University of Texas at Austin, 2013

Supervisor: Christine Julien

Parking garages have remained fairly outdated even as embedded systems have been introduced virtually everywhere to improve the human existence. Some provide information about whether they are full or not, but that does not offer a better parking experience, it only informs people once they are already there and is inconvenient. This causes people to circle the parking lot numerous times, making the process inefficient and wasteful. The SmartPark parking system fills that gap by providing an automated infrastructure that collects information regarding the availability of parking spaces in a garage. As modern technology grows and expands the connectivity available on automobiles, it would be even possible to interface with the car itself to provide parking information. Each space has an ultrasonic sensor attached to a microcontroller that communicates with a master, who keeps and displays the overall count of spaces available. The purpose of this paper is to provide the capability of dynamically adding and removing slaves, without requiring individual configuration for each slave prior to its deployment. A sequence of communication exchanges will be described in order for a slave to register itself with its master. Through a series of messages, the slave will be able to identify its location and begin reporting the state of its space, and the master will continue to keep track of existing slaves and their states. The result of the research is a protocol that allows successful pairing of a new slave with its master without previous

static configuration, which allows an easy deployment of the system without dependence on its original configuration. This functionality will make the system more scalable, allowing the parking system to be extended by connecting new slaves wherever they are needed. It will also make it more maintainable, since slave replacement or relocation will become an easy task. SmartPark can easily be adapted to existing parking structures with only the installation of the master and slave nodes, due to its limited resource requirements. Related work is also discussed and an insight into how this methodology can be used to modernize current automated parking systems is provided.

# Table of Contents

# List of Figures

## Chapter 1. Introduction

Smart parking systems are still not very common despite the prevalence of embedded systems in the automation of many aspects of everyday life. However, they are slowly appearing in locations where traffic is a concern and where the parking process needs to be made more efficient. Some intelligent parking systems provide information regarding the automobiles that have entered and left a garage, with simple sensors at the entrances that report current state [5]. More complex systems are starting to emerge as well, where individual sensors placed in a parking space send occupancy information to a central system that processes the data. These systems provide many types of user experiences, from simple displays with availability data to full guiding systems that will lead the driver to an empty spot [6]. These require very specific care, are not easily installed anywhere, and require detailed configuration and maintenance that cannot be done by someone without training.

SmartPark is an intelligent parking system that is similar to others in the market, where sensors are placed in every parking space and report to a master device. The devices in the system use a PIC32 microcontroller that is low cost and has all the peripherals necessary to control the different components in the slave and master devices. These microcontrollers manage the behavior of the devices; and require configuration that allows the slave devices to know which master should receive their information. However, because the microcontrollers are very versatile, the devices' implementation can be extended to provide functionality that other smart systems lack and that would be beneficial for the parking garage owners while improving the customer experience. A good example of this is the possibility of the system to recognize and configure new slave devices automatically without interaction from the user. The purpose of this paper is to explore a design for dynamically adding and removing master devices and slave sensors to the SmartPark automated parking system.

Having a system dynamically detecting when a slave device joins the system and locate their master to start reporting the state is important for several reasons. First of all, it is necessary to overcome the limitation caused by the requirement of specific configuration on slave devices. It facilitates the deployment of systems in any parking structure without high maintenance. The proposed design will allow the creation of a parking system that will require virtually zero configuration. Current implementations rely on a centralized system that identifies sensor devices in a parking lot and gathers information about the overall state. Even though they provide the desired functionality, the high maintenance and expertise required to service these systems do not allow them to become more prevalent. This barrier can be overcome by making them easier to install and care for, and the technology itself will no longer be an impediment for its dissemination. Once SmartPark is integrated into a parking structure, it will not require further interaction with specialized personnel. If a slave sensor or master device breaks, it can easily be replaced by connecting a new one in its place, instead of having to call a technician to do the job, since it may not be possible in every location. This will avoid the dependencies created when a system requires particular care, which can prevent someone from employing the system or even from the system being available in certain locations.

The dynamic element of the system can be achieved by implementing a mechanism that allows devices to identify their master and be recognized as part of the infrastructure. This scheme requires master and slave interfaces in the different devices, creating a tree structure that will allow every master to detect all its children, and the children to acknowledge their master. This design is simple enough for the transmission of state information, which can be known by the different masters at any point in the tree, and can be integrated into any type of parking structure without specific configuration requirements. It differs from other zero configuration [14] approaches in that it does not require an external networking infrastructure to provide any kind of services, or other network routing devices to delimit device domains. The design was created to be

incorporated into the current SmartPark implementation. However, it is generic enough that it can potentially be used in other sensor networks that do not require high throughput or constant communication. A simple case where this could also be applied is a storage facility with temperature sensors in every space, and the slave devices can inform their master whenever the temperature surpasses a threshold condition; or perhaps even inform the availability of a storage space to allow for easier administration. These systems are mostly for monitoring purposes, where a sensor indicates current state but do not require control operations or any other intelligent actions for the system.

## Chapter 2. Motivation

Many companies around the world are already dedicating resources to making their products more eco-friendly. Automobiles become more efficient everyday, and there are initiatives to make every aspect of a person's life more environmentally conscious. The time people spend driving around with the sole purpose of finding a place to park causes pollution and is a waste of time and resources. Some cities have already begun development towards a smarter parking system in dense urban areas where traffic is a concern, to minimize the time spent in the car looking for an available parking space. Implementations that provide access to parking information and allow drivers to easily find a space without complications are emerging and are an area of opportunity.

Not only are smart parking systems green, but they also contribute to a better quality of life. The amount of time a person spends in a car waiting in traffic and circling busy parking lots to find free spaces could be spent in other activities. Eliminating the waste of time allows people to dedicate this time to things they enjoy, affecting their behavior and their life in a positive way. Not only does this benefit drivers, but the effect can also be seen in the way a business operates. When a parking lot is close to its full capacity, it is generally quite difficult to find a space, and not only that, if there are no accurate ways to tally the parking spots, at some point the garage has to claim they are full when in reality there can be some spaces available. A smart parking system allows parking lots to operate at full capacity, without losing any business [4]. In the long run, these systems end up providing a better service to their customers, increasing their satisfaction, and consequently the establishment's profit.

There are some intelligent parking systems already in existence. Some of them only register vehicles that exit and enter the parking garage, which cannot locate available parking spaces even if they can somehow accurately tally them [5]. More complex systems have been developed as well, which use sensors on each space that rely

on a centralized master that gathers information [6]. This means that the individual sensors must be configured specifically to be in one location in order for the master to know where the information is coming from. Some systems require a detailed list of existing sensors, and their installation becomes cumbersome and fragile [5]. This paper focuses on the simplification of these types of systems based on the detachment of slave devices from a particular configuration, which is not a common trait in current implementations, since they are overly complex. Making a system dynamic without static device configuration allows flexibility upon installation and easy management during its lifetime.

There are also costs related to the sensors and electricity that allow the system to operate, so having a specialized technician assigned for system maintenance can become costly over time; because it requires a certain amount of expertise. Having a system that requires low maintenance and where each node can easily be replaced is important for scalability, not only within a single parking system, but deploying in multiple parking lots. Having a dynamic system without dependence on a trained individual can make this technology available for any parking garage owner and not only high-end establishments.

## Chapter 3. Related Work

There are multiple existing instances of intelligent parking systems that have been implemented around the world. Some of the most relevant are described in the following section, and their relation to SmartPark will be highlighted.

One of the largest implementations of smart parking systems in the US is described in [4]. The Baltimore Washington International Airport has an intelligent parking garage that guides drivers to vacant spaces. The installation has sensors in each space that report their state to the central management system, which keeps track of the current status of all the spots in the entire parking lot. It also has arrows in different sections of the building that guide customers to unoccupied areas where they can park. There is no tree structure within the system, and all the individual sensors depend on the central master.

A similar approach is implemented by CirPark[8], a system developed by a Spanish company that has deployed multiple smart parking systems across the world, including the parking structure at the Toulouse Blagnac Airport. This system is similar than the one described in this paper. There are ultrasonic sensors in every parking spot that report to a concentrator that sends its collected information to a centralized server in the network. This entity keeps information regarding the state of the parking system, and is also capable of setting specific configuration for the overall infrastructure. Each space also indicates whether it is taken or available with a red or green light, and has the capability of adding visual indicators for drivers to provide information about the areas they are passing. This system is very robust, and provides further features like electric charging stations and plate number tracking of the cars within the garage. The provider offers specialized software that manages all the sensors, concentrators, and other network devices that allows them to be easily configured. However, the configuration needs to be done for every space, and the system can be difficult to manage even with the available

tools. It is a viable solution that provides similar services to the ones described in this paper; however, it is still complex and not easily maintainable. The company that owns the system provides training for parking specialists that will need to install, configure and maintain the system. In comparison, SmartPark eliminates the need for a particular configuration, making the system more stand-alone without specialized care.

There has also been a recent surge of citywide proposals to create smart street parking. One important example of this is the San Francisco SFPark [5]. A wireless sensor is installed in each parking spot on the street, and they all report to wireless aggregators located in the city that forward information to a data warehouse. The data warehouse can also receive queries from an external API that is useful for applications in smart phones and is available to the public. The aim of the system is to facilitate the location of parking spots in the downtown and highly transited areas, but it also manages pricing based on demand depending on the location and time of day. The only intelligent feature of actual parking garages included in this proposition is the sensors installed on their gates to track how many cars enter and leave the facility. However, its street infrastructure is comparable to that of Smart Park, with the limitation that the sensors in SFPark require an inventory to be kept regarding their location that can be accessed by the single data store management system. This constraint is analogous to a static configuration for the sensors, since each of them has to be located and registered, causing maintainability issues that are addressed by SmartPark. There are other systems that locate street parking in cities like Nice and Barcelona or the LAExpressPark in Los Angeles, with similar limitations.

Lastly, there is another branch of smart parking where a robot system performs the entire parking process without intervention of the driver. Automation Parking [11] has developed automatic robotic systems as parking structures that fit a high amount of automobiles within a small space. It is made from an elevator system that takes the car and places it in a metallic cubicle, and can easily be retrieved in the same manner. Since

it is completely automated, it requires a complete infrastructure that cannot easily be integrated into existing structures. It is mostly used in New York City where space is very limited and optimization of its usage is necessary to accommodate the large amount of parking required. This is an example that fits a specific type of need, where parking is limited and creating a system that basically stacks cars can satisfy it. This intelligent system has its own merits, but cannot be deployed in existing garages like SmartPark can, and requires very specific maintenance and care.

## Chapter 4. System Architecture

As Figure 1 shows, the SmartPark system is composed of master and slave devices, each with a network enabled microprocessor. Each parking space has an ultrasonic range finder that detects when a car parks and when it leaves. This sensor is connected to a PIC32 microcontroller, and the communication is done via $I^2C$. This microcontroller acts as a slave in the system, relaying information regarding its state to a master aggregator only when the state changes. This master is another PIC32 configured to tally spaces based on the messages received from connected slaves, and it has the ability to display this data on an OLED screen. The slave microcontroller also controls two LEDs; the green LED is turned on to indicate the space is empty, and the red one is lighted to indicate an occupied spot.

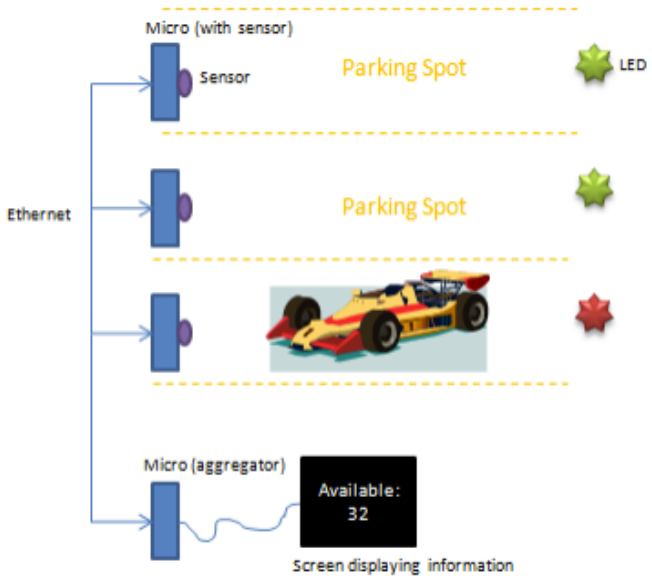


Figure 1. SmartPark overview

The master and slave components can be seen in Figure 2. Both master and slave devices have a PIC32 that controls the rest of the modules. The slave microcontroller

communicates with the range finder sensor through an $I^2C$ peripheral. The MaxSonar range finder [12] used to prototype the system employs a sound of high frequency to detect the presence of an object in a specified area, and it sends information about the distance of a target in front of it through its $I^2C$ interface. The slave microcontroller's LEDs are controlled with a pair of GPIO lines to indicate availability of the space. The master device on the other hand, uses its UART peripherals to communicate with a GSM module to reply to SMS message queries, and to display the current availability of its area through an OLED display. The flash memory in each node contains the entire configuration necessary for the devices to know their location and communicate with their appropriate masters. The slaves need to be configured with their master's IP in order for them to send messages when they are connected. The current configuration is static; and needs to be saved manually on each device. However, the following section will describe the methodology required for the slaves to be dynamically added or removed, and the system will detect their presence or absence and act accordingly.

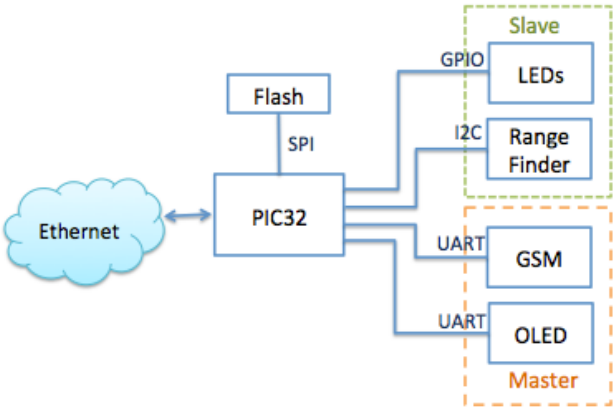


Figure 2. Smart Park system components

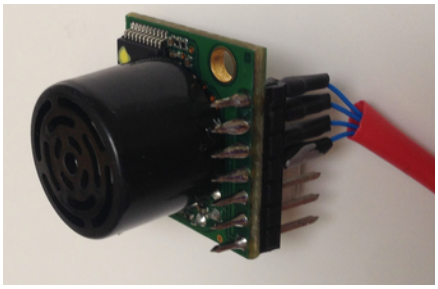

Figure 3. Range Finder Sensor

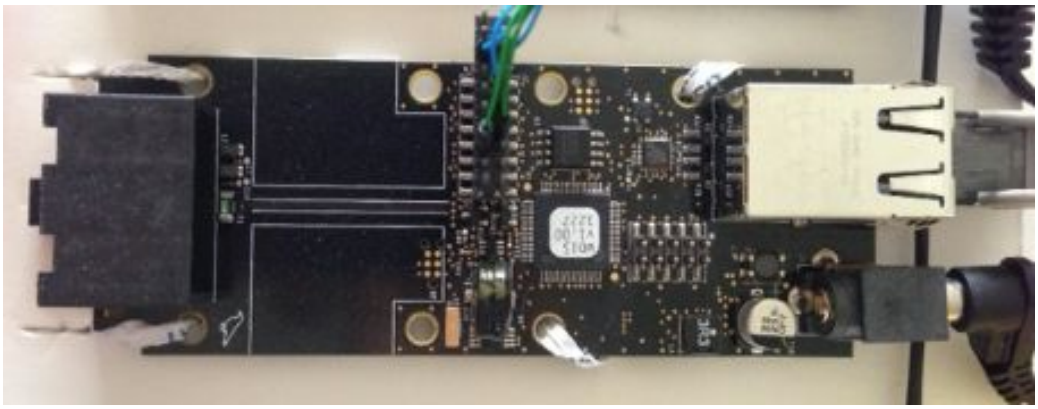

Figure 4. PIC32 Setup with Ethernet connection

The system can be extended to include a hierarchy of masters that can relay information to the above level, and the root master has the overall parking availability. In a typical parking garage, there can be several floors, each with multiple sections that can be again divided in smaller areas such as rows. In such a structure, each floor would have a master, with several masters per section, where multiple slaves would be located, one per parking spot. All the floor masters would report to the root master, which would control the main parking display; and would reply to SMS queries regarding the current overall availability.

# Chapter 5. Dynamic Slave Support Design

This section will describe the protocol that will allow a slave with no prior configuration to be added to the system, to be configured by a master, and to begin reporting its state. The methodology also defines a process that allows the masters to know if a slave is present or left the system, and to aggregate the information from their known slaves. The first part details a simple design that is limited by only having two layers of masters in the network tree. The second part describes a more complex structure that can accommodate multiple masters in any configuration, which mimics a real application in existing parking buildings.

## 5.1 BASIC DESIGN

The simplest configuration for the master and slave nodes in the parking system would be a hierarchical tree, where the root is a master aggregator node, its children other master nodes, and their children the actual sensor slaves. An example of this as an Ethernet network is shown in Figure 5. In this configuration, the master aggregator has a static IP that is known by all the other masters in the system. Each master is in its own subnetwork, allowing the broadcast messages to remain in a single parking section.

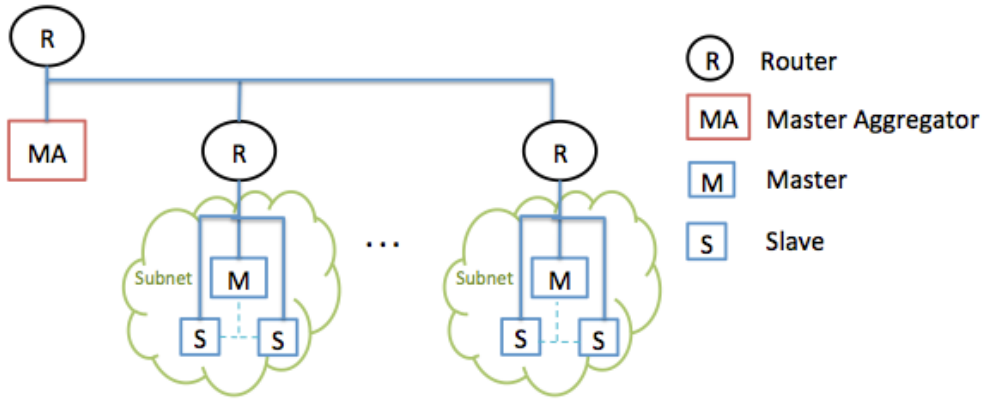


Figure 5. Basic design diagram

The dynamic nature of the system and the zero configuration condition on the slave nodes requires the use of a broadcast message to announce newly introduced sensor nodes. Upon connection, a slave node broadcasts a CONNECT message containing its IP. The master node in that sub-network receives the message, saves the new IP in its slave table, and replies with its own IP. Once the slave receives the reply, it saves its master configuration. The slave is now part of the system, and every update regarding the state of its space will be sent to that master.

Every master collects information of all the slave sensors within its subnet; and sends periodic updates to the master aggregator. Based on the information presented, it could be argued that there is no need for the local subnet masters, since the IP of the slaves is information enough for the aggregator to know their location. The slaves could be preconfigured to send their messages to the master aggregator, and the structure of the system would be significantly simplified.

The master aggregator now has all the data for the entire parking garage; and can provide the overall availability per location. Every section has a screen displaying the

availability within that specific area, and the information presented on the display is controlled by the master aggregator.

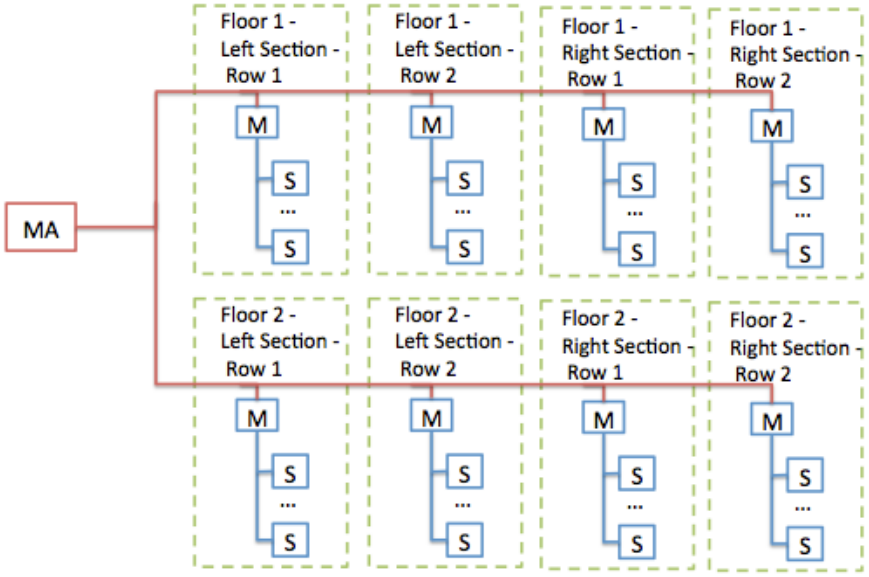


Figure 6. Basic design applied to a parking structure

This is a simple approach that introduces a dynamic element to the system; however, it has multiple disadvantages. Broadcast messages are being used, which limits the depth of the network tree. In this configuration, a master cannot be another master's slave, since a single subnet would contain two masters. In such a case, the CONNECT broadcast message from a new slave would reach both of them, and there is no information in the slave message that would help the masters decide who it should belong to. Such a decision would depend on the section the new slave is located, and without specific configuration, the slave cannot provide it. This is why only one master per subnet is allowed, and each subnet can represent a section in the garage. Another drawback is the dependency of all the displays on the master aggregator. In theory, it seems acceptable to have it control all the displays, but in reality, as observed above, a

parking garage with multiple rows, sections and floors would require a more complex hierarchy that is not feasible with this configuration.

### 5.2 GENERALIZED DESIGN (TWO INTERFACE MASTERS)

As mentioned in the previous section, having a single master aggregator does not allow for scalability and ease of use. This section discusses a network tree configuration that allows a master to have slave masters, allowing a deeper hierarchy that is more adaptable to many different parking structures. A root master is the top tier of the tree, and it will aggregate all the information acquired from the entire system. The masters in the lower levels propagate information upwards; creating a chain of aggregated information that is collected at different stages, and can be displayed at multiple points, depending on the parking lot design.

The following diagram displays the arrangement of the nodes in this system. The masters have two interfaces, an uplink that talks to parent masters, and a downlink that talks to its slaves. These two interfaces allow new slaves to connect to the master that owns the segment, requiring no preexisting configuration on the device. The slaves only have uplinks to communicate with their masters. Only a single master per section is allowed, separating each trunk in the tree where many masters can exist as slaves, but only one master will act as such and respond to messages. A segment is the bus connection between the master downlink and its slave uplinks. A root master will be the only one without an uplink connection; therefore, a master can identify itself as the root master.

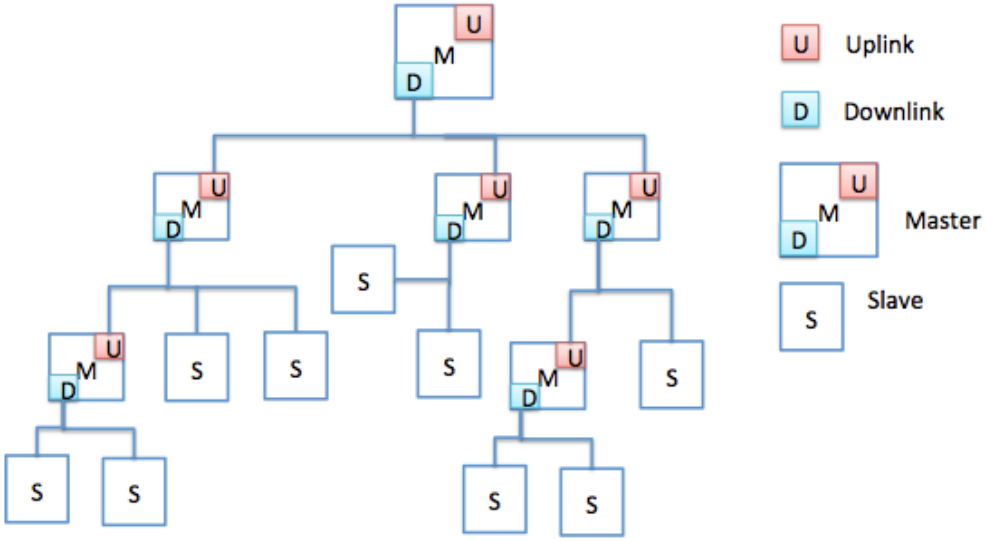


Figure 7. Diagram of the generalized design

The root master and slave masters can be selected and set depending on how the parking garage is arranged. A good example is shown in the figure below. The root master has one master slave on each floor section, each of which has master slaves for each row, and those have sensor slaves. This can be further extended if the floors are too large, where a master per floor can be selected, and then each section can be divided as mentioned previously. This also permits the parking designers to set displays in different areas that can display each section's space availability without the root master's intervention. The root master will control the main display that can be exposed to outside potential customers, and it will also reply to SMS requests. Slave devices within the different parking sections will have no knowledge about their location, so it is not necessary for them to associate to a specific parking spot in the garage. Each section will be delimited by master devices that will display the availability data corresponding to their branch. Because slaves report to a master, and masters indicate section availability, a driver can navigate through the parking garage, following the displayed information

based on free spaces. A driver can be guided starting from broad sections like floors, to specific areas in that floor and all the way to a particular row.

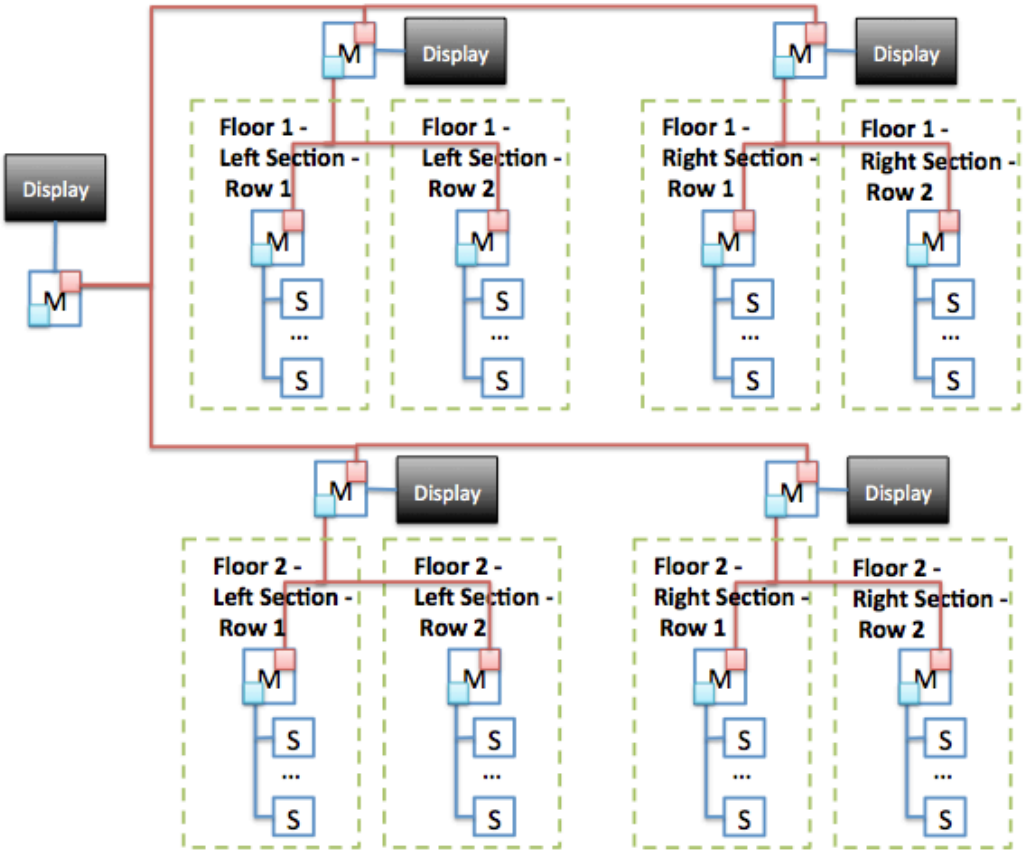


Figure 8. Generalized design integrated into parking garage

This design demands an architecture that can limit the broadcast domain, which brings Ethernet to mind. However, the design is protocol agnostic, since it has no other requirements that would force this standard to be used. The bus can be Ethernet, but it can use other protocols like CAN or RS-485, since there are no particular bandwidth or speed requirements, and the messages are short and sporadic. In the case of an Ethernet connection between devices, a DHCP server can provide IPs in each segment of the tree,

a function that is commonly performed by routers in the network. The devices can then communicate using broadcasts if they don't know the master's IP, and can send directed messages once dynamic configuration is achieved. There are other options that can be explored when this protocol is used, since the design does not require an IP to be used as an ID. Therefore, communications can be done in the data link layer, which would make the physical infrastructure cheaper.

If the Controller Area Network standard was used in the system, a CAN bus would connect all the devices in a single branch of the tree. The framework does not require high throughput so the 1 Mbps maximum that CAN offers would suffice. Since CAN provides a multi-master bus, all nodes connected to it can broadcast messages, which fulfills the design's requirements. Communication would be done through CAN data frames, where the identifiers could be the individual device's serial number, which would be unique. The masters would always have an identifier of 1 on their downlink, to ensure the highest priority. Access to the bus is done based on priority, so it is true that in this configuration, devices with lower serial numbers would get higher priority. However, the number and frequency of transmissions required is not significant for this to be detrimental. Also, the Carrier Sense Multiple Access bus arbitration allows one node to transmit at a time, which is acceptable due to the fact that slaves only transmit information when the state of their space changes or when they attempt to connect.

Another option that can be explored would be the use of the RS-485 standard, which describes the physical connection that allows a multi-drop network where multiple devices, masters and slaves, can exchange data. Since the communication protocol can be defined depending on the implementation, addressing can be handled by the segment masters. A master device can assign an address to the slaves upon connection, and a broadcast address can also be defined which can be used by all slaves without special configuration. Even though a single device can transmit at a time, as it was mentioned with CAN, it is not a disadvantage. In fact, this is a possibility in any implementation

using a microcontroller that has serial peripherals, which can be interfaced to converters that can provide TS-485 connectivity.

### 5.2.1 Dynamic slave detection

When a slave node is added to a given section of the tree, it broadcasts a CONNECT message with its own id. The term id will refer to the identification number used during transfers, whether it is a unique identifier such as a serial number or a network address such as an IP. The master of the segment receives it and sends a reply with its own id, which is stored by the slave. This allows the slave nodes to send updates regarding the state of their parking space. If the device being added is a master in slave mode, then it also sends a CONNECT message, and it contains not only its id but also that it is a master node. The segment master then replies to it in the same way, and the master slave will save its master's id and send its own aggregated information in message updates. A CONNECT message is always followed by an UPDATE to indicate the initial state of the slave space or section availability. The master of the segment will receive updates from a single space or from a section that has been aggregated by a master in slave mode, and will keep track of all the available spots in the whole branch beneath it.

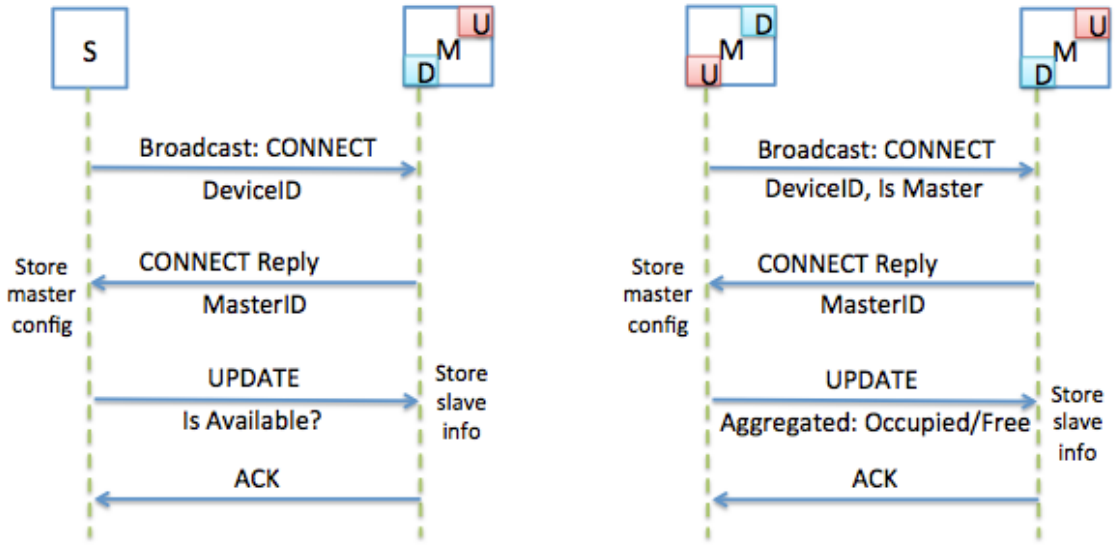


Figure 9. Message exchanges between master and slave upon connection

For a master to consider its slaves as present, it needs to receive a keepalive every two seconds. This particular period of time can vary, but two seconds is a reasonable period of time to avoid flooding the bus but still have an accurate picture of the active devices. If a slave has been removed from the system, meaning that the master does not receive the keepalive, then it is removed from the master's slave table. Removing a slave from the master's table causes the space to be removed from possible availability, whether the last message indicated that the space was free or not. The keepalives also add a level of reliability, since the master does not keep non-current information. It is possible that a slave gets mistakenly disconnected from the system, whether it is due to a power outage or a cable being removed by accident; in this situation, the slave maintains its configuration, and will send a RECONNECT message to its configured master upon connection, which will contain the state of its parking spot or section if it is also a master. The master will re-add the slave to its table and update its aggregated space count. Every UPDATE message will be acknowledged by the master, which will indicate to the slave that the master is present. In the case where the master does not reply, the slave will try

to send the update a defined amount of times, and if it does not get a reply, it will switch to send a RECONNECT during defined intervals for a period of time to try to identify its master. If a master is not found, the slave device will begin broadcasting a CONNECT message, allowing master devices to be replaced as well. A slave's configuration is saved in its flash, therefore it is not volatile; however, it is possible to reset the hardware with a switch to manually erase its entire saved configuration. If a slave is connected with old configuration, a RECONNECT without a reply will be sent a defined number of times, and if a master does not send a response, a CONNECT message will be broadcast to update the current configuration.

This methodology allows a slave to be removed form the system, re-added, and moved. Old slave sensors can be relocated, and new ones can be added in case the parking garage is expanded or if old slaves are broken or obsolete. Since the slaves require no previous configuration, no special training or expertise is required to setup the system. Therefore, scalability is no longer an issue, adding new slaves, or even whole new segments with new masters, is not problematic.

## Chapter 6. Integration into SmartPark

The design to support dynamically adding and removing slave devices can be easily integrated into the existing SmartPark system. The above design does not require the Ethernet standard to be used; however, the current operation of SmartPark is done using the protocol, and therefore, the implementation will be done with Ethernet. The following section will describe two different approaches that can be taken to introduce the desired functionality. The first will describe a method that would require minimal effort, and the second transforms the current configuration to use less resources and make it more efficient.

The figure below depicts the setup of a tree branch within the system for the first approach. Each segment in the tree has a router that acts as a DCHP server, which assigns an IP to every slave upon connection, allowing it to begin sending broadcasts within the subnetwork and communicating with its future master.

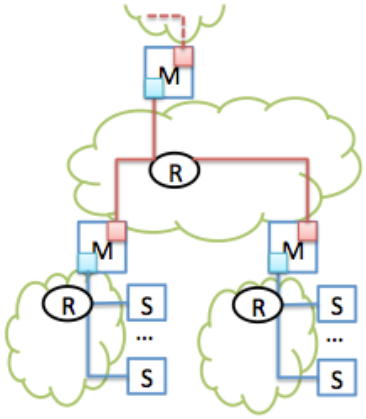


Figure 10. Diagram of the extended SmartPark design

This configuration allows the uplinks and downlinks of the masters to be isolated, and limits the reach of the slave broadcast, as mentioned in the basic design. This

implementation is only one option that requires the least amount of modifications, which is done relying on the routers' functionality, such as their DHCP servers. Currently, the individual slave devices have static IPs, so their master's configuration stored in the flash never changes, and each sensor slave must be preconfigured in order for it to report its state. The new implementation eliminates the need for static IP addressing, and the slave makes a DHCP request to the router upon connection. Once the slave has its IP, it then broadcasts a CONNECT message containing its IP, and the message exchange as described in the previous chapter follows. In this case, the IP is the id that will identify both slave and master to each other, and will allow the message to reach its destination.

However, it is possible to reduce the complexity of the system deployment by integrating this functionality into the master devices. The benefit of doing this is reducing cost by removing the additional requirement of having a router per segment. In a small scale, it may not seem significant, but when the design is applied to a parking system that could have several branches, the cost does add up. By incorporating the routing capabilities to the master devices, the need for external routers would be eliminated, but the complexity of the system would increase significantly. It would be technically possible to do this, but a different and simpler methodology will be described.

A different approach would be to remove the need for the DHCP server, and the need for routing, since all communication will be done inside each segment. The design described in the previous chapter does not require any routing capabilities; in fact, it only requires the ability to send broadcast messages. This can be done in the link layer, so an IP is not even necessary. The messages exchanged are encapsulated within an Ethernet frame, with unique source and destination MAC addresses. The contents of a CONNECT message are shown in the following figure.

| Destination Address | Source Address | Type | Data |
|---|---|---|---|
| 6 bytes | 6 bytes | 2-bytes | 14 bytes |
| -FF-FF-FF-FF-FF | Slave's MAC Address | X | "CONNECT_S" or "CONNECT_M" followed by 0xFF values for stuffing |

Figure 11. Contents of the CONNECT message

The data in the message for a slave sensor node is CONNECT_S, while the data for a master node connected through its slave interface is CONNECT_M. The master receives the message, and because the MAC address corresponds to a broadcast, the information contained in the data field is accepted and decoded. The master does not need any further information in the frame, since it will not be sent to layers above the link layer. Once the master classifies the message as a CONNECT, it replies with its own MAC address with a frame that contains the slave's MAC as the destination address. When the slave receives the reply, it updates its flash configuration with its master information and begins sending UPDATE messages. Any further communication will be directed to the master of the segment, and its MAC will be set as the destination address.

| Destination Address | Source Address | Type | Data |
|---|---|---|---|
| 6 bytes | 6 bytes | 2-bytes | 14 bytes<br>"UPDATE_00_00_00" |
| Master's MAC Address | Slave's MAC Address | X | *Byte 8* will contain a 0 or 1 for slave sensors, and 'M' for master slave devices<br>*Bytes 10-11* will contain the occupied spaces on master slave devices<br>*Bytes 13-14* will contain the available spaces on master slave devices |

Figure 12. Contents of the UPDATE message

Figures 11 and 12 illustrate the contents required for the different types of message transmissions. The CONNECT and UPDATE messages require specific details in the data section that will indicate whether a device is a sensor slave or a master in slave mode. The RECONNECT message is the same as the CONNECT message, with "RECONNECT_M" and "RECONNECT_S" as the content data to distinguish one from the other. Any other message, including keepalives and ACKs will be sent with their respective string in the content section of the message.

### 6.1 OVERALL IMPLEMENTATION DETAILS

The current SmartPark implementation is based on the Microchip TCP/IP stack [13], and TCP clients and servers are used to exchange messages between masters and slaves. All the slave devices have a static IP and their master IP saved as part of their configuration. In order to remove the dependency on the network layer, all the TCP communication needs to be replaced by lower layer message transmissions. Instead of using the IP to send and receive messages, the devices' MAC addresses are used to determine the message's source and destination.

The TCP/IP stack provides functions to easily open and close TCP connections as well as to send data through them. However, lower layer functions exist, even though they are not readily exposed to the user. There are multiple functions that the TCP/IP stack uses to access the media and send its buffers. This set of MAC functions allows for the creation of data frames that can be directly transmitted through Ethernet, and the functionality to receive data in the same way is also provided. The functions MACPutHeader() and MACGetHeader() handle the source and destination MAC addresses required as parameters for the message, as well as setting a type for the message. Furthermore, functions for writing and retrieving data to the Ethernet buffer are also provided, which can be used to directly interact with the link layer without going through the network layer.

An example of the code required to send and receive data using the MAC addresses only is shown below:

```
// Send message
while(!MACIsTxReady());
MACSetWritePtr(BASE_TX_ADDR);
MACPutHeader(&rcvMacAddr, MAC_IP, sizeof(ackArray));
MACPutArray((BYTE*)&ackArray, sizeof(ackArray));
MACFlush();

// Receive message
while (!MACGetHeader(&rcvMacAddr, &type));
MACGetArray((BYTE*)&receiveArray, sizeof(receiveArray));
```

These code snippets demonstrate the capabilities provided by the stack, and can be an advantage for this particular implementation. For the master device, a thread

listens for connection messages, and sends replies with its own MAC address. In the case of the slave device, whether it is on it a sensor slave or a master slave uplink interface, a thread checks if there is an existing master configuration, and if there is none, it sends CONNECT messages and waits for a reply. Another thread exists for both devices. The master's downlink interface is constantly listening for any communication from a device, which can be an UPDATE or a KEEPALIVE, and if applicable, maintains the display up to date. The slave sensor device is constantly checking if there is a change in the state of its parking spot, and sends an UPDATE message if appropriate.

## 6.2 RESULTS

The dynamic system design was prototyped to verify that the functionality described in this paper could be successfully integrated into the existing SmartPark system. The framework used will be described, as well as the observed outcome of the implementation.

### 6.2.1 System Setup

The design was prototyped using a scaled version of a parking garage. There are currently no hardware prototypes that have two Ethernet interfaces available, so multiple single interface devices were used to prototype a single segment within a system. A slave microcontroller with its range finder was used as a sensor slave, connected to the bus through its interface. The same microcontroller was repurposed to appear as a master slave device, to simulate an uplink interface. A master device was connected to the bus to represent the only downlink in the segment, acting as an overall master. This allowed the functionality of all the major components of the system in a single tree branch to be tested with the dynamic component added to the system.

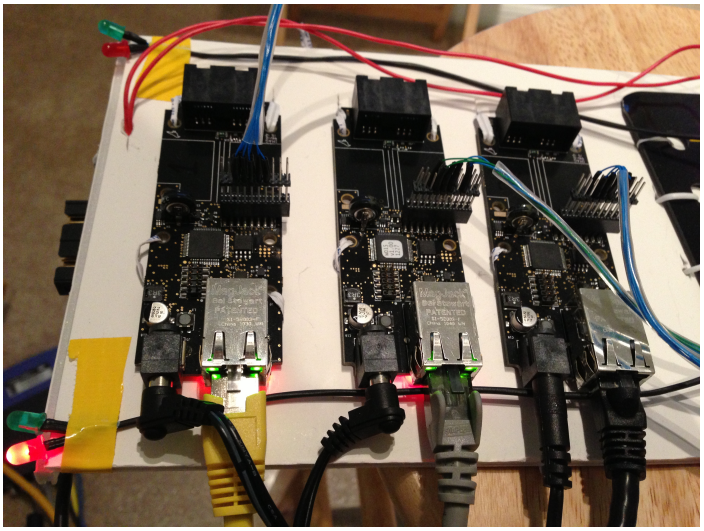

Figure 13. Microcontroller system setup


Figure 14. OLED Display indicating available spaces in the system

### 6.2.2 System Behavior

Once each device was programmed to perform its role within the simulated segment, a series of tests were performed. Both master slave and sensor slave devices were connected to the bus without having any previous configuration. They sent their CONNECT messages, which were observed using the Wireshark [15] tool on a separate computer connected to the Ethernet bus. The slave configuration was successful in both cases, and the devices proceeded to send their UPDATE messages to inform the master of their current state. The master slave device was configured to send its UPDATE based on its sensor input; therefore, whenever the sensor indicated that the space had been vacated or occupied, the master sent an UPDATE_0M with a defined amount of available and occupied spaces, which was handled by the segment master and displayed appropriately.

The three microcontrollers that were part of the scaled system were reprogrammed to perform each role within the system, to vary the MAC addresses of each master and slaves. This validated that the slave's configuration was updated upon connection, and that the devices could interchange roles without problems. Because a slave does not respond to CONNECT messages, even if an intended RECONNECT message is sent due to an existing old configuration in a device, the missing reply causes the slave to broadcast a new CONNECT message and update its existing configuration.

## Chapter 7. Future Work

Additional features can be implemented for the SmartPark project that can expand its functionality and make it more appealing. It is possible to include an alternate wireless setup, to add more configuration capabilities to the masters, to add a cloud computing element to the system, or to provide integration to public parking informational services. The following section will discuss these potential improvements.

A substitute for the existing wired design would be a setup in which the slave and master devices communicate through a wireless network. This would require a redesign of the devices to use a radio for communication, which is not trivial. Furthermore, adding a wireless component to the current design would involve not only a change in the hardware, but also in the way the slave devices communicate and associate to their master. A consequence of the lack of bus delimitation for a particular master domain would be the need for a new protocol for the dynamic behavior of the system. However, removing the dependence on a wired infrastructure would make these devices easier to set up, and potentially reduce installation costs. Therefore, it could be advantageous to explore this alternative design, and reconsider the design proposed in this paper in favor of a more wireless friendly scheme that could make the SmartPark system even more versatile.

Currently, the system does not provide a way to interactively configure more information regarding the location of the master. It could be possible to assign a section name to it for easier administration. One solution would be to provide the root master a list of names that will be assigned to the rest of the masters. This solution would require access to the root master to provide the list of assigned names linked to the specific serial numbers or MAC addresses of the masters that will be connected to the system. Another possible implementation would take advantage of the Ethernet connection used in SmartPark, in which all masters could be configured initially with a default static IP, a

DHCP server and a DNS server. With this setup, when a computer is connected to the master, it will be assigned an IP and any webpage loaded will be redirected to the device's configuration page. This approach could extend the configuration performed on a master, and it could contain any information such as name of the area, maximum amount of parking spots and location within the parking structure.

Cloud computing is something that would add extra features to the system, rather than extending or modifying current behavior like the ones described above. SmartPark could provide a service where clients can send data about their parking garage systems, for it to be stored, accessed and analyzed. This data could include statistics regarding the parking lot usage during specific periods of time, which can be useful for business model analysis. The information could shed light regarding peak times and low traffic times, which can be used to adjust prices based on time and day, something the SFPark system already does. Another advantage of this service would be for companies that have multiple parking garages in different geographic locations, since they can localize information about all of them in one place and generate knowledge that can be applied to their administration and future parking lot sites. Reports about slave devices, both sensors and masters, can also be gathered and can be useful for tracking failures and creating maintenance plans. This cloud service would allow all SmartPark customers to access information about their systems, which can be helpful to grow and manage their organizations.

A further extension of the capabilities described above would be to not only have an information server available to the owners of the parking lots, but to have other data available to potential users of the garage. Currently, SmartPark responds to SMS messages indicating the total number of spaces available at the time of the query. However, there are existing services that provide information about parking in certain geographic locations. Smart phone applications are very common these days, and they are used to indicate parking locations in selected areas without sending a text message or

even without knowing about the existence of the parking lot. Since the system already knows about overall availability, it would be possible to add internet connectivity to the root master so that it could communicate with these third party services. Adding the capability for SmartPark to publish its information or respond to outside web queries would increase the garage's exposure and grow their customer base.

## Chapter 8. Conclusion

The constant evolution of automated parking systems aiming to provide a better service and higher occupancy rates has created an opportunity for intelligent systems to gain a higher presence in existing parking garages. The SmartPark system was created to provide a smart solution that allows constant monitoring of the availability of spaces in a parking lot. The design presented in this paper accomplished the pairing of master and slave devices so that the slave sensors can start reporting their state to their assigned master automatically, without requiring individual configuration about their current spot. This feature sets the SmartPark system apart from other similar existing technologies. The dynamic aspect of the system allows fast and easy deployment, but more importantly, it permits its maintenance without technical expertise. This will allow the technology to spread without boundaries, having no ties to the service provider after installation. Furthermore, the SmartPark system can easily be extended to include more features and have other differentiators that will be appealing to potential customers. The master and slave devices have microcontrollers can be easily programmed to include these features without sacrificing the dynamic nature of the system.

## Bibliography

[1] Chen, Hanxing, and Jun Tian. "Research on Controller Area Network", *Proceedings of the International Conference on Networking and Digital Society*, Guiyang, May 2009, pp. 251-54.

[2] Li, Bai. et al. "Research on the detecting system of distributed nodes based on RS-485 bus", *Proceedings of the 2010 International Conference on Educational and Network Technology*, Quihuangdao, June 2010, pp. 422-25.

[3] Castro, M. et al. "Well-Known Serial Buses for Distributed Control of Backup Power Plants. R8-485 versus Controller Area Network (CAN) Solutions", *Proceedings of the IEEE 28th Annual Conference of the Industrial Electronics Society*, November 2002, pp. 2381-86, Vol. 3.

[4] Charette, Robert M. "Smart Parking Systems Make It Easier to Find a Parking Space", October 2007. <http://spectrum.ieee.org/green-tech/advanced-cars/smart-parking-systems-make-it-easier-to-find-a-parking-space>.

[5] San Francisco Municipal Transportation Agency. "SFPark: Putting Theory Into Practice", August 2011. Print.

[6] Ross, Valerie. "Smart Parking Systems Steer Drivers to Open Spaces", *Popular Mechanics*. February 16, 2011. Web.

[7] Kurose, J. et al. *Computer Networking, A Top-Down Approach.* Addison Wesley. Fifth Edition, 2010.

[8] Jose, D'Opazo. "CirPark: Efficient & Green Parking Concept", January 2013. <http://www.circontrol.com/docs/EFFICIENT_AND_GREEN_PARKING_CONCEPT_2013.pdf >.

[9] Shaheen, Susan A, Caroline J. Rodier and Amanda M. Eaken. "Smart Parking Management Field Test: A Bay Area Rapid Transit (BART) District Parking Demonstration", University of California, Berkley, January 2005.

[10] Zhang, Liping, et al. "SafeTrip-21 Connected Traveler: Networked Traveler Transit and Smart Parking", University of California, Berkley, March 2011.

[11] Automotion Parking Systems. *Car Parking: UP System.* New York, June 2013. <http://www.automotionparking.com/downloads/pdf/Car_Parking_UP_System.pdf>.

[12] MaxBotics Incorporated. *I2CXL-MaxSonar – EZ Series. High Performance Range Finder Datasheet.* 2005.

[13] Rajbharti, Nilesh. *The Microchip TCP/IP Stack.* Microchip Technology Inc, 2002.

[14] Hong, Se Gi, Suman Srinivasan and Henning Shulzrinne, "Accelerating Device Discovery in Ad-Hoc Zero Configuration Networking", *Proceedings of the Global Communications Conference*, Washington, November 2007, pp. 961-65.

[15] Wireshark. <http://www.wireshark.org/>.